TR_1D_model1_SS\TR_1D_model1_func_calc_A_int.m

```
% TR_1D_model1_SS\TR_1D_model1_func_calc_A_int.m
%
% function [A_int,iflag] = ...
%    TR_1D_model1_func_calc_A_int(...
%       x_state,epsilon,Param);
%
% This function calculates the A matrix components
% in the general DAE form
%
%   epsilon(k) * df_dt(k) = b(k)
%       - sum_{j} {A(k,j)*x_state(j)}
%
% for the interior points in a 1-D model of a
% tubular reactor that is discretized by upwind
% finite differences.  It is written to be compatible
% with the use of a non-uniform grid.  x_state is a
% column vector of the concentrations of each species
% and the temperature.  If the number of grid points
% is Grid.num_pts, then the length of x_state
% and the dimension of the square matrix A is
%
%   num_DOF = Grid.num_pts *
%       (ProbDim.num_species + 1)
%
% The A matrix is calculated using general routines
% that return square matrices of dimension
% Grid.num_pts that discretize at the interior points
% the first and second derivatives respectively.
% These routines are written to perform the
% discretization only at those points corresponding
% to non-zero values in an integer mask - the use of
% the epsilon section corresponding to the first
% field performs the discretization at all interior
% points.  Then, a second generic shift function
% takes these matrices and returns their values in
% the appropriate locations in A based on the number
% of the field (either ifield in 1 to num_species for
% a concentration field or ifield = num_species + 1
% for the temperature field).  These matrices are
% added one by one, multiplied by the appropriate
% physical constants, to build the A matrix for
% the problem.
%
% INPUT :
% =======
% x_state             REAL(num_DOF)
%                   this is a column vector containing
```

```
%                  the state data
% epsilon             INT(num_DOF)
%                  this is a 1-D array of integers of the same size
%                  as x_state.  It contains a 1 at position k for
%                  every equation k that is an ordinary differential
%                  equation, and a 0 for every algebraic equation,
%                  which in this problem arise from the boundary
%                  conditions
% Param               this is a data structure used to
%                    pass the structures :
%                    ProbDim, Reactor, Physical, Grid
%
% OUTPUT :
% ========
% A_int               REAL(num_DOF,num_DOF)
%                    This is the matrix that
%                    discretizes the transport
%                    terms at the interior points.
% iflag               INT
%                    if negative, then error
%                    if 0, routine did not complete
%                    if 1, successful operation
%
% Kenneth Beers
% Massachusetts Institute of Technology
% Department of Chemical Engineering
% 7/2/2001
%
% Version as of 7/25/2001



function [A_int,iflag] = ...
   TR_1D_model1_func_calc_A_int(...
      x_state,epsilon,Param);

iflag = 0;

func_name = 'TR_1D_model1_func_calc_A_int';


% This flag controls what action to take in
% the case of an assertion error or the
% failure of a called subroutine to execute
% properly.
i_error = 2;


% First, extact parameter structures.

if(~isstruct(Param))
```

```
   iflag = -1;
   message = [func_name, ': ', ...
      'Param is not a structure'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      A_int = 0;
      return;
   end
end


% ProbDim
if(~isfield(Param,'ProbDim'))
   iflag = -1;
   message = [func_name, ': ', ...
         'Param does not contain ProbDim'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      A_int = 0;
      return;
   end
end
ProbDim = Param.ProbDim;

% Reactor
if(~isfield(Param,'Reactor'))
   iflag = -1;
   message = [func_name, ': ', ...
         'Param does not contain Reactor'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      A_int = 0;
      return;
   end
end
Reactor = Param.Reactor;

% Physical
if(~isfield(Param,'Physical'))
```

```
   iflag = -1;
   message = [func_name, ': ', ...
         'Param does not contain Physical'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      A_int = 0;
      return;
   end
end
Physical = Param.Physical;

% Grid
if(~isfield(Param,'Grid'))
   iflag = -1;
   message = [func_name, ': ', ...
         'Param does contain Grid'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      A_int = 0;
      return;
   end
end
Grid = Param.Grid;


% calculate the number of state variables
num_DOF = (ProbDim.num_species+1)*Grid.num_pts;

% check x_state
dim = num_DOF; check_column = 1;
check_real = 1; check_sign = 0; check_int = 0;
assert_vector(i_error,x_state,'x_state',func_name, ...
   dim,check_real,check_sign, ...
   check_int,check_column);

% check epsilon
dim = num_DOF; check_column = 1;
check_real = 1; check_sign = 2; check_int = 1;
assert_vector(i_error,epsilon,'epsilon', func_name, ...
   dim,check_real,check_sign, ...
   check_int,check_column);
```

```
%PDL> Allocate A_int and initialize to zeros

max_nonzero = 3*num_DOF;
A_int = spalloc(num_DOF,num_DOF,max_nonzero);



%PROCEDURE: FinDiff_1D_FirstDeriv
%PDL> Calculate a square matrix of dimension
%   ProbDim.num_pts that discretizes in 1-D the first
%   derivative operator at each interior point by
%   upwind finite differences on a non-uniform grid.
%ENDPROCEDURE

% create integer mask of interior points
imask_int = linspace(1,1,Grid.num_pts)';
imask_int(1) = 0;
imask_int(Grid.num_pts) = 0;

iuse_upwind = 1;

[FirstDerivMatrix,iflag_func] = FinDiff_1D_FirstDeriv( ...
   Grid,imask_int,iuse_upwind,Reactor.velocity);
if(iflag_func <= 0)
   iflag = -3;
   message = [func_name, ': ', ...
         'Error (',int2str(iflag_func), ') ', ...
         'returned from FinDiff_1D_FirstDeriv'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      return;
   end
end


%PDL> For every concentration and temperature field,
%   add to A the contribution from the convective
%   flux term.
%        FOR ifield FROM 1 TO ProbDim.num_species + 1

num_fields = ProbDim.num_species + 1;

for ifield = 1:num_fields


%        PDL> Increment A by the product of
%       Reactor.velocity multiplied by a matrix
```

```
%      returned by the generic shifting function
%          PROCEDURE(shift_discretization_matrix)
%      that shifts the dimension Grid.num_pts
%        matrix obtained by discretizing the
%      first derivative matrix to the appropriate
%      spot in A for field number ifield.  This
%      is required because the A matrix is written
%      to be used with the master state vector that
%      is a column vector first of the concentrations
%      of species 1 at each point, then of species 2,
%      etc., then the temperature field.  This shifting
%      procedure may be used for any geometry
%      (2D, 3D, etc.) as long as the same format is
%      used to stack the fields sequentially in the
%      master state vector.

   [A_shifted,iflag_func] = ...
      shift_discretization_matrix( ...
      Grid.num_pts,num_fields,FirstDerivMatrix,ifield);
   if(iflag_func <= 0)
      iflag = -4;
      message = [func_name, ': ', ...
         'Error (', int2str(iflag_func), ') ', ...
         'returned from shift_discretization_matrix'];
      if(i_error ~= 0)
         if(i_error > 1)
            save dump_error.mat;
         end
         error(message);
      else
         return;
      end
   end

   A_int = A_int + Reactor.velocity*A_shifted;


%PDL> ENDFOR

end


%PROCEDURE: FinDiff_1D_SecondDeriv
%PDL> Calculate a square matrix of dimension
%   ProbDim.num_pts that discretizes in 1-D the second
%   derivative operator at each interior point using
%   central differences on a non-uniform grid.
%ENDPROCEDURE

[SecondDerivMatrix,iflag_func] = ...
   FinDiff_1D_SecondDeriv(Grid,imask_int);
```

```
if(iflag_func <= 0)
   iflag = -5;
   message = [func_name, ': ', ...
         'Error (', int2str(iflag_func), ') ', ...
         'returned from FinDiff_1D_SecondDeriv'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      return;
   end
end


%PDL> For every concentration field, add to A the
%   contribution from the diffusion flux term
%        FOR ifield FROM 1 TO ProbDim.num_species

for ifield = 1:ProbDim.num_species


%        PDL> Increment A by the product of the negative of
%   Physical.diffusivity(ifield) times the matrix
%   obtained from PROCEDURE(shift_discretization_matrix)
%   applied to the dimension num_pts discretization matrix of
%   the second derivative called with the value of ifield.

  [A_shifted,iflag_func] = ...
      shift_discretization_matrix( ...
      Grid.num_pts,num_fields,SecondDerivMatrix,ifield);
   if(iflag_func <= 0)
      iflag = -6;
      message = [func_name, ': ', ...
         'Error (', int2str(iflag_func), ') ', ...
         'returned from shift_discretization_matrix'];
      if(i_error ~= 0)
         if(i_error > 1)
            save dump_error.mat;
         end
         error(message);
      else
         return;
      end
   end

   A_int = A_int - Physical.diffusivity(ifield)*A_shifted;


%PDL> ENDFOR
```

```
end


%PDL> Now, add to A the appropriate matrix for the thermal
%   conduction term.  Tnen increment A by the negative of
%   the product of the thermal diffusivity and the matrix
%   obtained from PROCEDURE(shift_discretization_matrix)
%   applied to the dimension num_pts discretization matrix
%   of the second derivative called with
%   ifield = ProbDim.num_species + 1.

ifield = ProbDim.num_species + 1;

[A_shifted,iflag_func] = ...
   shift_discretization_matrix( ...
   Grid.num_pts,num_fields,SecondDerivMatrix,ifield);
if(iflag_func <= 0)
   iflag = -7;
   message = [func_name, ': ', ...
         'Error (', int2str(iflag_func), ') ', ...
         'returned from shift_discretization_matrix'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      return;
   end
end

A_int = A_int - Physical.thermal_diff * A_shifted;


iflag = 1;

return;
```